NASA/CR-1999-209551
ICASE Report No. 99-36

Parallelization of a Multigrid Incompressible Viscous Cavity Flow Solver Using OpenMP

Kevin Roe and Piyush Mehrotra
ICASE, Hampton, Virginia

Institute for Computer Applications in Science and Engineering
NASA Langley Research Center
Hampton, VA

Operated by Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

Prepared for Langley Research Center
under Contract NAS1-97046

September 1999

PARALLELIZATION OF A MULTIGRID INCOMPRESSIBLE VISCOUS CAVITY FLOW SOLVER USING OPENMP*

KEVIN ROE AND PIYUSH MEHROTRA

Abstract. We describe a multigrid scheme for solving the viscous incompressible driven cavity problem that has been parallelized using OpenMP. The incremental parallelization allowed by OpenMP was of great help during the parallelization process. Results show good parallel efficiencies for reasonable problem sizes on an SGI Origin 2000. Since OpenMP allowed us to specify the number of threads (and in turn processors) at runtime, we were able to improve performance when solving on smaller/coarser meshes. This was accomplished by giving each processor a more reasonable amount of work rather than having many processors work on very small segments of the data (and thereby adding significant overhead).

Key words. parallel computing, SGI Origin 2000, OpenMP, multigrid, viscous driven cavity

Subject classification. Computer Science

1. Introduction. Effective use of parallel machines requires easily maintainable and portable programming models that allow users to exploit parallelism in applications written in a standard high-level language. MPI provides portability, however it can be more difficult to maintain and is not a high-level programming model. High Performance Fortran (HPF) is portable and fairly easy to maintain. OpenMP is also portable on shared memory architectures and fairly easy to maintain, although it can only be used on shared memory machines [1, 2, 3]. Since HPF can be used on both shared and distributed memory machines, one would think that it would be the parallel programming model of choice. However, OpenMP has some advantages over HPF if one is using a shared memory machine. Most notably, the OpenMP model allows users to incrementally parallelize their code, which can be invaluable when parallelizing large codes.

Viscous fluid flow inside a driven cavity is a popular test case for evaluating numerical techniques. We describe the parallelization of a multigrid scheme for solving the viscous incompressible driven cavity problem using OpenMP. Two-dimensional incompressible viscous driven cavity flows are computed using a loosely coupled, implicit, second-order centrally differenced scheme. Mesh sequencing and a three-level V-cycle multigrid solver with a simple Jacobi integration smoother are used in this study. Although a multi-level V-cycle would more likely appear in practice, the three-level V-cycle was used simply to test OpenMP's performance.

In this paper we explain the viscous driven cavity problem, the governing equations and the numerical methods used. We then discuss the strategy used to parallelize the code. Tests were conducted to determine the performance of OpenMP when an equal number of processors are used for each grid level, and to see if overhead could be removed by using less processors for solving on coarser grids. Finally we discuss possible future work involving additional tests for the multigrid algorithm when more than three grid levels are allowed, as well as possible performance comparisons between OpenMP, MPI and HPF.

2. Problem Domain. A *driven cavity* is defined as a rectangular cavity (the cases in this paper are all done on square cavities) where a plate of infinite length moves across the top of the cavity from left to

*This research was supported by the National Aeronautics and Space Administration under NASA Contract No. NAS1-97046 while the authors were in residence at the Institute for Computer Applications in Science and Engineering (ICASE), NASA Langley Research Center, Mail Stop 132C, Hampton, Virginia 23681-2199. {kproe,pm}@icase.edu.

right with non-dimensional speed U (Figure 2.1). The domain is discretized on a two-dimensional structured Cartesian mesh. Although the code was written to allow for clustering of mesh points near the walls, the tests conducted were all done on meshes with uniform cell spacing.

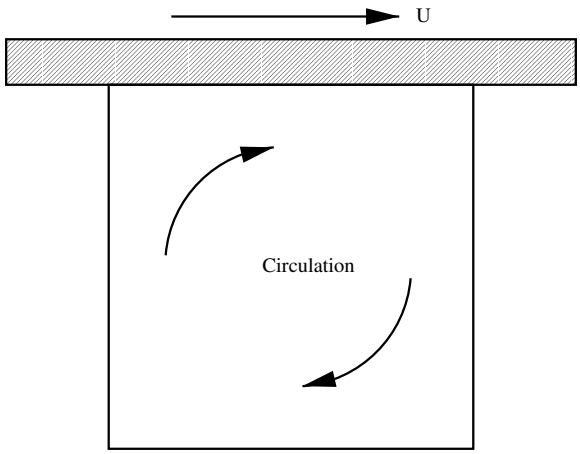

FIG. 2.1. *Square cavity with an infinitely long plate moving across the top of the cavity*

3. Governing Equations. The non-conservative form of the governing equations for unsteady incompressible laminar viscous flow is as follows

$$u_x + v_y = 0, \tag{3.1}$$

$$u_t + uu_x + vu_y + P_x = \frac{1}{R_e}\nabla^2 u, \tag{3.2}$$

$$v_t + uv_x + vv_y + P_x = \frac{1}{R_e}\nabla^2 v, \tag{3.3}$$

where R_e is the Reynolds number.

An alternative method to solving the governing equations can avoid the explicit appearance of pressure by using the vorticity and stream function as dependent variables in the two-dimensional case. The two-dimensional vorticity transport equation is as follows

$$\zeta_t + u\zeta_x + v\zeta_y = \frac{1}{R_e}\nabla^2 \zeta, \tag{3.4}$$

where ζ is the vorticity defined by

$$\zeta = u_y - v_x. \tag{3.5}$$

The stream function in two dimensions is defined by

$$u = \psi_y \text{ and } v = -\psi_x. \tag{3.6}$$

Substituting this into (3.5) produces the Poisson equation for the stream function

$$\nabla^2 \psi = \zeta. \tag{3.7}$$

The governing equations (Equations 3.1-3.7) are discretized using second-order expressions for spatial derivatives and a first-order implicit expression in time. The equations can be rewritten in two simplified forms:

$$f_t + uf_x + vf_y - \frac{1}{R_e}\nabla^2 f = -g \tag{3.8}$$

$$-\nabla^2 f = -g \tag{3.9}$$

where,

$$f = [u, v, \zeta, P, \psi], \tag{3.10}$$

$$g = [P_x, P_y, 0, gg, \zeta] \text{ and} \tag{3.11}$$

$$-gg = (u_x + v_y)_t + u_x^2 + 2u_y v_x + v_y^2 - \frac{1}{R_e}\nabla^2 (u_x + v_y). \tag{3.12}$$

We create a loosely-coupled set of equations by evaluating f implicitly at the advanced time step, while the non-linear terms u, v, and g are lagged at the previous time step. The resulting discretization is as follows:

$$\frac{f - f^n}{\Delta t} + \frac{u^n}{2\Delta x}(f_{i+1} - f_{i-1}) + \frac{v^n}{2\Delta y}(f_{j+1} - f_{j-1}) - \frac{1}{R_e}\left[\frac{1}{\Delta x^2}(f_{i+1} - 2f + f_{i-1})\frac{1}{\Delta y^2}(f_{j+1} - 2f + f_{j-1})\right] = -g' \tag{3.13}$$

and,

$$-\frac{1}{\Delta x^2}(f_{i+1} - 2f + f_{i-1}) - \frac{1}{\Delta y^2}(f_{j+1} - 2f + f_{j-1}) = -g' \tag{3.14}$$

where f represents f^{n+1}. g' is the discretized form of g,

$$g' = \begin{bmatrix} \frac{P_{i+1}^n - P_{i-1}^n}{2\Delta x} \\ \\ \frac{P_{j+1}^n - P_{j-1}^n}{2\Delta y} \\ \\ 0 \\ gg' \\ \zeta^n \end{bmatrix} \tag{3.15}$$

where,

$$-gg' = -\frac{(u_x + v_y)}{\Delta t} + \frac{u_{i+1}^n - u_{i-1}^n}{4\Delta x^2} + \frac{(u_{j+1}^n - u_{j-1}^n)(v_{i+1}^n - v_{i-1}^n)}{2\Delta x \Delta y} + \frac{(v_{j+1}^n - v_{j-1}^n)^2}{4\Delta y^2} - \frac{1}{R_e}\left[\frac{(u_x + v_y)_{i+1}^n - 2(u_x + v_y)^n + (u_x + v_y)_{i-1}^n}{\Delta x^2} + \frac{(u_x + v_y)_{j+1}^n - 2(u_x + v_y)^n + (u_x + v_y)_{j-1}^m}{\Delta y^2}\right]. \tag{3.16}$$

Also note that

$$(u_x + v_y)^{n+1} = 0 \text{ and} \tag{3.17}$$

$$(u_x + v_y)^n = \frac{u_{i+1}^n - u_{i-1}^n}{2\Delta x} + \frac{v_{j+1}^n - v_{j-1}^n}{2\Delta y.} \tag{3.18}$$

We can rearrange Equations 3.13 and 3.14 into

$$A_{i,j}f_{i,j} + A_{i+1,j}f_{i+1,j} + A_{i-1,j}f_{i-1,j} + A_{i,j+1}f_{i,j+1} + A_{i,j-1}f_{i,j-1} = B_{i,j}. \tag{3.19}$$

When we are solving for u, v, and ζ, the coefficients are defined as:

$$\begin{aligned} A_{i,j} &= 1 + 2\frac{\Delta t}{R_e \Delta x^2} + 2\frac{\Delta t}{R_e \Delta y^2}, \\ A_{i\pm1,j} &= -\frac{\Delta t}{R_e \Delta x^2} \pm \frac{\Delta t}{2\Delta x}, \\ A_{i,j\pm1} &= -\frac{\Delta t}{R_e \Delta y^2} \pm \frac{\Delta t}{2\Delta y} \text{ and} \\ B_{i,j} &= f^n - \Delta t\, g'. \end{aligned}$$

When we are solving for P and ψ, the coefficients are defined as:

$$\begin{aligned} A_{i,j} &= \frac{2}{\delta x^2} + \frac{2}{\Delta y^2}, \\ A_{i\pm1,j} &= -\frac{1}{\Delta x^2}, \\ A_{i,j\pm1} &= -\frac{1}{\Delta y^2}, \\ B_{i,j} &= -g'. \end{aligned}$$

Now we can solve the linearized system of governing equations in the form of $Af = b$. A more in depth discussion on the solution of the viscous driven cavity problem can be seen in other references [5, 6, 7].

4. Numerical Methods. The flow solver approximately inverts a penta-diagonal linear system at each time step in an attempt to have the dependent variables reach a steady state condition. A Symmetric Gauss-Seidel (SGS) iteration is used to approximately invert $Af = b$. A V-cycle multigrid is used to more quickly damp low frequency errors from the solution; however, the V-cycle is currently limited to three grid levels (fine, medium, and coarse). In addition, grid sequencing is used initially to speed the convergence of the solution on the finest grid. A more detailed description of the multigrid algorithm can be seen in other references [8, 9].

5. Parallelization Strategy. Even though the multigrid code used here was initially written without any thought of being parallelized, only a few changes to the code were required prior to its parallelization. Initially, a Symmetric Gauss Seidel (SGS) algorithm was used to speed convergence; however, there were complications when attempting to parallelize this algorithm; a red-black version of this algorithm was found to be unstable [4]. Similarly, when either a regular or red-black Gauss Seidel algorithm was substituted in place of the SGS method, the solution became unstable. Thus, a simple Jacobi algorithm was inserted to replace the SGS algorithm for testing purposes.

Aside from this, modifications to the code were only in the form of parallel directives specifying the number of processors to use in parallel sections, which loops to parallelize, and which variables should be considered shared or private. The code was incrementally parallelized, which aided tremendously in the conversion process for two reasons. First, we did not have to determine the data mapping for all the variables

in the entire code. Second, using OpenMP in contrast to MPI or HPF made it easier to efficiently parallelize sections of the code and determine bottlenecks.

The next test involved using a different number of processors for each grid level. Although we assign all available processors when computing at the finest grid level, this may not yield a sufficient amount of work for each processor when computing at the coarse grid level. In fact, the synchronization overhead, when using all processors at a coarser grid level, will most likely degrade overall performance. Since it is in shared memory, we do not have to worry about the cost associated with reshaping the data if we wish to use less threads/processors (than at the fine grid level) in OpenMP, we can assign a different number of processors to each grid level in the multigrid algorithm to better optimize performance (Figure 5.1). This code was written so that we can specify the number of processors to use for each grid level in an input file. Thus the code can be run with a different number of processors per grid level without recompilation.

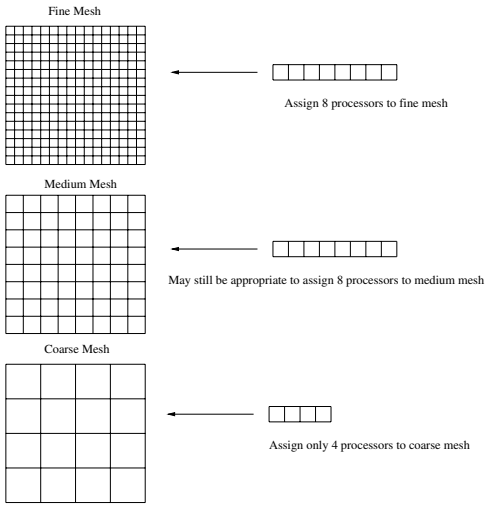

FIG. 5.1. *Example: Employing a different number of processors for each mesh level*

6. Results. The viscous driven cavity flow solver was tested on the SGI Origin 2000 at NASA Ames Research Center. All timings were measured using clock calls within the code rather than profiling in order to remove as much overhead as possible; also, results were the average of ten runs. Timings were started after initial I/O and stopped when the solution had converged. The code was compiled with the *MIPSpro 7.30 f90* compiler using OpenMP directives.

Results show sufficient parallel efficiencies when reasonable problem sizes were tested on the SGI Origin 2000. Table 6.1 and Figure 6.1 show that when the coarse grid size is sufficiently large (a factor of 8 smaller than the fine grid) we are able to achieve good parallel efficiencies. When the fine grid is 64x64, the medium grid is only 32x32 and the coarse grid is only 16x16. Because there is little work to do for each processor, it is difficult to obtain much benefit from using multiple processors. When the fine grid is increased to 256x256 (the medium grid is 128x128 and the coarse grid is 64x64), there is now more work for each processor to do even when multiple processors are used. For the 256x256 grid, two processors yield an excellent parallel efficiency of 0.94 and then steadily drops off as more processors are assigned. Larger problems continue this trend obtaining a parallel efficiency of 0.96 for two processors in the 512x512 fine grid case. Also, notice that when the code was parallelized with OpenMP, it maintained good single node performance. This is an advantage over HPF since many current HPF compilers are still having difficulty obtaining good single node

TABLE 6.1
Performance of driven cavity problem using OpenMP

Processors	Fine Grid Size							
	Execution Time (sec)				Parallel Efficiencies			
	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512
seq	6.62	32.60	193.35	4963.34				
1	6.81	33.10	198.82	4990.18	0.97	0.98	0.97	0.99
2	4.18	18.52	102.85	2582.37	0.79	0.88	0.94	0.96
4	3.09	11.13	56.21	1375.71	0.54	0.73	0.86	0.90
8	2.21	8.41	35.90	817.85	0.37	0.48	0.67	0.76
16	1.97	5.26	23.59	512.76	0.21	0.39	0.51	0.60

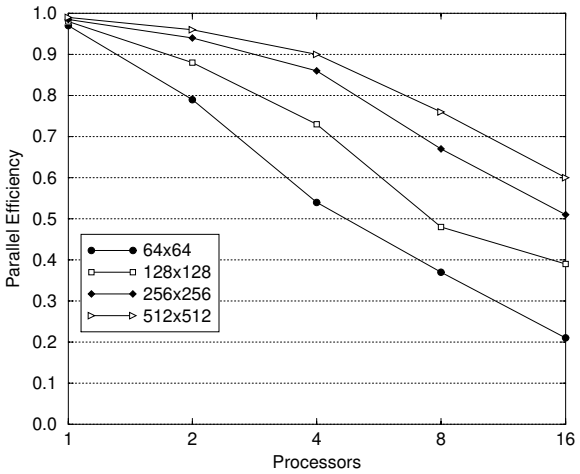

FIG. 6.1. *Performance of driven cavity problem using OpenMP*

performance.

Initially, we believed that the performance was lower than expected because of the Origin 2000's memory placement policy [10, 11]. This is because the arrays were initialized serially; thus the arrays were assigned to the processor which performed the initialization rather than being distributed among all the processors which subsequently operated on the arrays. The array was then initialized in parallel and although this improved performance, it was to a limited degree. If only a few iterations were run, then the impact on performance would be significant. When the code was run until convergence (on the order of 1000-10000 iterations depending on the test conditions) the impact on performance was small enough to be within the margin of error for our timings.

OpenMP also enables the user to specify the number of threads (and, hence processors) for each multigrid level, if desired. This allowed for an optimal number of processors to be specified at each grid level. Table 6.2 and Figure 6.2 show that execution time can be reduced when tuning the number of processors for each grid level. Although some benefit was obtained, it was not as major an improvement as originally hoped for.

Plots of the solution to the viscous driven cavity for $R_e = 1000$ using a 128x128 mesh can be seen in Figure 6.3. The streamline trace shows the main circulation zone close to the center of the cavity and two recirculation zones in the bottom corners of the cavity. The velocity vector plot also shows the main

TABLE 6.2
Performance with a different number of processors set per grid level

Processors			Fine Grid Size							
			Execution Time (sec)				Parallel Efficiencies			
fine	med	coarse	64x64	128x128	256x256	512x512	64x64	128x128	256x256	512x512
8	8	8	2.21	8.41	35.90	817.85	0.37	0.48	0.67	0.76
8	8	4	2.01	7.90	33.87	810.10	0.41	0.52	0.71	0.76
16	16	16	1.97	5.26	23.59	512.76	0.21	0.39	0.51	0.60
16	16	8	1.52	4.71	21.43	498.96	0.27	0.43	0.56	0.62

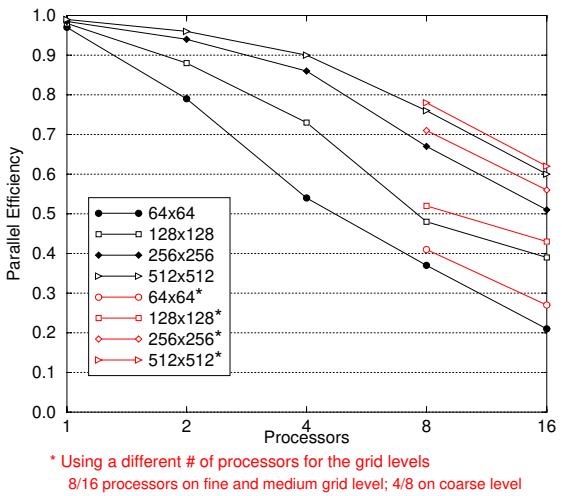

FIG. 6.2. *Performance when a different number of processors are specified for each multigrid level*

circulation zone as well as the two recirculation zones, although they are more difficult to see because of the differences in magnitude.

7. Conclusions/Future Work. The results show that OpenMP can achieve sufficient parallel performance with simple multigrid codes. The code was written such that the user can specify the number of processors for each grid level in the configuration file so that re-compilation is not necessary. It is conceivable that a routine could be incorporated into the code that would determine the optimal number of threads to use for each grid level at runtime. The next stage would be to remove the three level restriction and re-test OpenMP's parallel performance. This would most likely show even more of a reduction in overhead when very coarse grids are used (relative to the fine grid). A direct comparison to MPI and HPF equivalent codes would also be desired, however it would be difficult to compare the use of a variable number of processors for each grid level since there would be implementation difficulties when using MPI or HPF (specifically with reshaping of the data at runtime which would be prohibitively expensive).

REFERENCES

[1] E. AYGUADE, M. GONZALEZ, J. LABARTA, AND X. MARTORELL, *Multi-level and Dynamic Parallelism Exploitation in OpenMP,* DAC/CEPBA Technical Report, Computer Architecture Department, Polytechnic University of Catalunya, 1998.

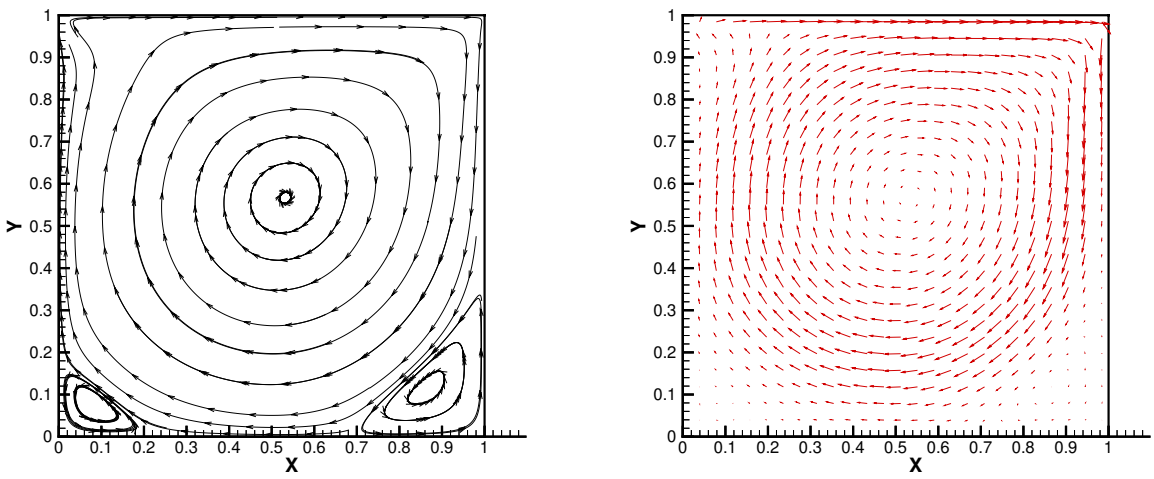

FIG. 6.3. *Streamline Trace and Velocity Vector Plot for* $R_e = 1000$ *using a 128x128 Mesh*

[2] *OpenMP: A Proposed Industry Standard API for Shared Memory Programming.* www.openmp.org, October 1997.

[3] *OpenMP Fortran Application Program Interface.* Version 1.0, www.openmp.org, October 1997.

[4] G. GOLUB AND J. ORTEGA, *Scientific Computing: An Introduction with Parallel Computing.* Academic Press, Inc., 1993, pp. 340-350, 387-388.

[5] H. NISHIDA AND N. SATOFUKA, *Higher-order Solutions of Square Driven Cavity Flow Using a Variable-order Multi-grid Method*, International Journal for Numerical Methods in Engineering **34**, 1997, pp. 637-653.

[6] K.A. HOFFMANN AND S.T. CHIANG, *Computational Fluid Dynamics For Engineers - Volume I*, Engineering Education System, 1993.

[7] W.A. WOOD, *Multigrid Approach to Incompressible Viscous Cavity Flows*, NASA TM 110262, 1996.

[8] W.L. BRIGGS, *A Multigrid Tutorial*, Society for Industrial and Applied Mathematics, 1994.

[9] P. WESSELING, *Introduction to multigrid methods*, ICASE Report No. 95-11, 1995.

[10] D.L. SONDAK AND J. PERRY *A Study of Memory Placement on an SGI Origin 2000*, Proceedings of the 9th SIAM Conference on Parallel Processing, 1999.

[11] SILICON GRAPHICS, INC. *Origin 2000 and Onyx2 Performance Tuning and Optimization Guide*, Document Number: 007-3430-002, 1998.